XFlow: An Extensible Tool for Empirical Analysis of Software Systems Evolution

Francisco Santana[1], Gustavo A. Oliva[2], Cleidson R. B. de Souza[3], Marco A. Gerosa[2]

[1]Faculdade de Computação – Universidade Federal do Pará (UFPA)
CEP: 66075-110 – Belém – PA – Brasil

[2]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
CEP: 05508-090 – São Paulo – SP – Brasil

[3]IBM Research – Brasil CEP: 04007-05 – São Paulo – SP – Brasil

wertherjr@gmail.com, {goliva,gerosa}@ime.usp.br,
cleidson.desouza@acm.org

Abstract. *Studies of software evolution have long tried to understand the interplay between the software itself and its developers. Such studies usually require extensive tool support due to large and complex data that need to be collected, processed, and analyzed. Although several tools have been proposed, they often provide limited support to studying software evolution by considering technical and social aspects in isolation. In this paper, we describe XFlow, an extensible tool for empirical analysis of software evolution. We start by presenting the tool features and its architecture, and then we show usage scenarios that support the formulation of empirical hypothesis based on insightful visualizations.*

1. Introduction

The highly complex global society imposes constant pressure for changes in software systems. The great amount of investment and the need for frequent end-user satisfaction leaves no room for software to be largely reworked each time a requirement changes. Indeed, software that is not tolerant to modifications is doomed to abandonment or replacement [Mens & Demeyer, 2008][Bode 2009]. Therefore, it is crucial that software systems can be easily adapted to continuous changes and flexible enough for the addition of new features to remain useful and to maintain business value [Bode 2009]. Such ability of software systems is known as evolvability.

The evolutionary process that software systems undergo has become a subject of great interest for both industry and academic communities and has motivated different studies. Several of these studies aim to empirically evaluate hypotheses related to the evolution of software and to the identification of software engineering best practices based on the analysis of how long-term successful projects maintain their status [Bennet & Rajlich 2000]. At the same time, several tools have been developed to support researchers when conducting such studies. Nevertheless, most of these tools focuses on either technical [Zimmerman *et al.* 2005][D'Ambros & Lanza 2009][D'Ambros & Lanza 2010] or social aspects [Gilbert & Karahalios 2007] of the software evolution process. Although there are a small number of tools that have incorporated both aspects [de Souza *et al.* 2005][Sarma *et al.* 2009], they provide limited support to help one investigate the relationship between the technical and social aspects. This poses a serious threat to the validity of these works, since previous works have noted that there is a close relationship between the software structure and the social structure of the organization involved in its development [Conway 1968][Parnas 1972].

In this paper we present XFlow, an extensible and interactive tool aimed to provide a comprehensive analysis of the evolution of software projects by mining software repositories. This is done taking into account both social and technical aspects of the software system. By bringing together these two views, we expect to support exploratory case studies that call for a deeper understanding of software evolution aspects such as: software architecture decay over time; the effects exercised by architectural decisions over the software project; developers' participation level, role changes, and knowledge about the system. In this sense, the tool can be employed to assist in the formulation and test of hypotheses. Hence, we show illustrative scenarios involving FLOSS systems where the proposed tool was employed to aid in the investigation of (i) how teams deal with the departure of key people, and (ii) how new features are accommodated in the software.

XFlow was built inspired in our earlier prototype tool, TransFlow [Costa *et al.* 2009], and took advantage of all lessons learned during the development and use of the latter. In particular, XFlow's architecture was designed to improve performance and address new requirements that arose while conducting empirical studies in open source software development [*ibid*.]. XFlow collects data from version control systems, then identifies dependencies and evaluates metrics over the collected data, and finally presents interactive visualizations depicting the entire software project chronologically.

The rest of this paper is organized as follows. In Section 2, we introduce the basic concepts that comprise the foundation of XFlow. Section 3 presents the tool design rationale and its visualization schemes. In Section 4, we show usage scenarios that highlight the tool ability to provide insightful information in exploratory case studies. In Section 5 we discuss the related work and, finally, in Section 6 we state our conclusions and plans for future work.

2. Background

Software repositories contain a plethora of information about the underlying software and the associated development process. Mining Software Repositories (MSR) is the research area concerned with the analysis of large amounts of data available in software repositories to uncover valuable information about software systems and projects.

In the following subsections, we briefly introduce the MSR area and situate XFlow within it. Afterwards, we describe the concepts of Logical Dependencies [Ball *et al.* 1996] and coordination requirements [Cataldo *et al.* 2006], both of which have inspired the design and construction of XFlow.

2.1.Mining Software Repositories

Mining Software Repositories (MSR) is a research topic in Software Engineering that focuses on the application of Data Mining techniques to software projects' historical data to provide a better understanding and support predictions about the software itself and its context [Ball *et al.* 1996][Koch, 2007][Olatunji *et al.* 2010]. For instance, MSR can aid in enhancing software development processes, indicating a developer's special proficiency and locating change patterns throughout source code files [Zimmerman *et al.* 2005]. Software repositories include: source code version control systems (VCSs), bug-tracking systems, and communication archives (e.g., discussion forums and emails) [Kagdi *et al*. 2007].

Kadgi *et al.* (2007) created a four-layered taxonomy to characterize MSR approaches in the context of software evolution. The authors claim that such taxonomy is sufficiently expressive and effective. The top layer (*software evolution*) regards the typical objects of study in MSR investigations: change characteristics of the high-level properties of a software system, detailed changes in the artifacts themselves, or both. The next layer (*purpose*) concerns the type of questions answered by such studies: market-basket questions (*if A occurs, then what else occurs on a regular basis?*) and prevalence questions. Instances of prevalence questions include metrics (e.g., how many times was method 'foo' modified?) and boolean queries (e.g., was method 'foo' removed from class A?). The subsequent layer (*representation*) refers to the type (e.g., physical), granularity (e.g., system, files, classes), and expression of the artifacts and their differences. Finally, the bottom layer (*information sources*) depicts the elements that are naturally available in software repositories and those that are derived and need to be made available to support MSR studies.

Regarding the first layer, XFlow supports the analysis of both software high-level properties and detailed changes in artifacts. In a similar fashion, XFlow comprises both market-basket questions (e.g., detection of logical coupling, see next section) and prevalence questions (e.g., lines of code added or removed per file in a commit). Although XFlow mines Subversion (SVN), which is limited to a physical-level representation of source code (i.e., file and line numbers), it is able to represent information from sources in terms of files and developers. Finally, in relation to the bottom layer, XFlow operates on source code and metadata obtained from commit log messages.

2.2. Logical Dependencies and Coordination Requirements

A dependency is a semantic relationship that indicates that a client element (usually compilation units, such as classes and packages) may be affected by changes performed in a supplier element [Booch 2007]. Logical dependencies are implicit dependencies between software artifacts that evolved together and, thus, are linked to each other, although not necessarily structurally related [Ball *et al.* 1996], [Gall *et al.* 1998]. These artifacts are connected from an evolutionary point of view, i.e., they have often changed together in the past, so they are likely to change together in the future. In a nutshell, logical dependencies occur whenever two artifacts are frequently co-changed. Unlike structural dependencies, this technique can spot dependencies between any kind of artifact that composes the system, including configuration files (such as XML and property files) and documentation.

Using the idea of logical dependencies, [Cataldo *et al.* 2006] proposed the concept of coordination requirements (CRs) to denote the relationship between the technical dependencies (which comprises both structural and logical dependencies) of a software system and the structure of the development work to construct such system. CRs are expressed in a developer-by-developer matrix that represents the extent to which pair of developers needs to coordinate their work. XFlow supports the calculation of the CR measure by implementing this approach.

3. XFlow

XFlow is an extensible tool developed entirely in Java whose main goal is to support empirical software evolution analyses by considering both social and technical aspects. By supporting both aspects, the tool aims to assist researchers in formulating and testing hypotheses by observing the evolution of different software projects.

3.1. Design Rationale

Focus on supporting empirical research. Even though XFlow can be used for several purposes (e.g., improve developers awareness or suggest changes to classes), the focus on supporting research in Software Evolution was set as a fundamental project decision.

Do not discard or repeat anything previously done. The tool is able to stop and resume each processing stage at will, allowing one to increase the amount of information available for a determined project as the project itself grows over time. XFlow also enables the observation of previously collected data from different perspectives by the application of different filters on the original data or by applying different statistical support and confidence rules [Zimmerman *et al.* 2005].

Keep it light. As previous related work has stated [Bevan *et al.* 2005], software repositories offer several challenges on data mining due to the large amount of computational resources required to handle them. To address such problems, XFlow counts on a light structure to map dependencies that not only requires less memory to operate, but is also efficiently represented on Database Management Systems (DMSs).

Flexibility as a key to achieve completeness. While designing XFlow, several questions (e.g., which coupling detection method is more efficient? which visual layout is more appropriate to represent dependencies between entities?) were raised. Since we believe that there is no absolute answer to these questions, one of XFlow's main features is its architectural extensibility, which enables new modules to be easily *plugged* in the tool. XFlow's flexible architecture (Figure 1) offers developers hooks to implement new functionalities with little or no impact on already implemented modules.

Filter and customize. Since researchers can easily be overwhelmed by the sheer amount of data regarding the evolution of a software project, another feature of XFlow consists in providing users with the ability to analyze only a subset of the data. In other words, the tool enables the use of multiple filters at different stages during the processing of the project. These filters limit data collection to contributions between specified dates or revisions or to specific repository folders or source code artifacts. It is also possible to select specific visualizations or metrics.

3.2. Visualizations

Currently, there are five different visualizations available on XFlow associated with a set of filtering mechanisms and controls. This plurality of options enables the visualization of software evolution aspects according to different perspectives and levels of abstraction, thus aiding in software comprehension and complexity management. In particular, by following Ben Shneiderman's visualization mantra "*overview first, zoom and filter, then details on demand*" [Shneiderman 1996], we designed XFlow to handle complexity and overcome some of the obstacles that pose difficulties and hinders empirical research in software evolution.

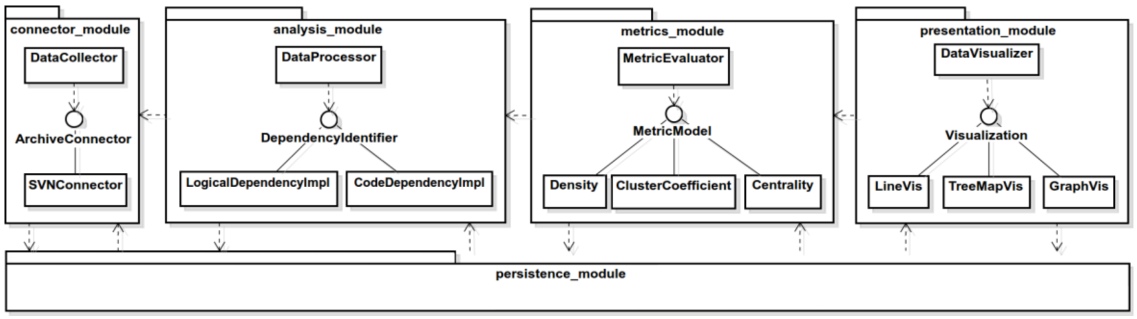
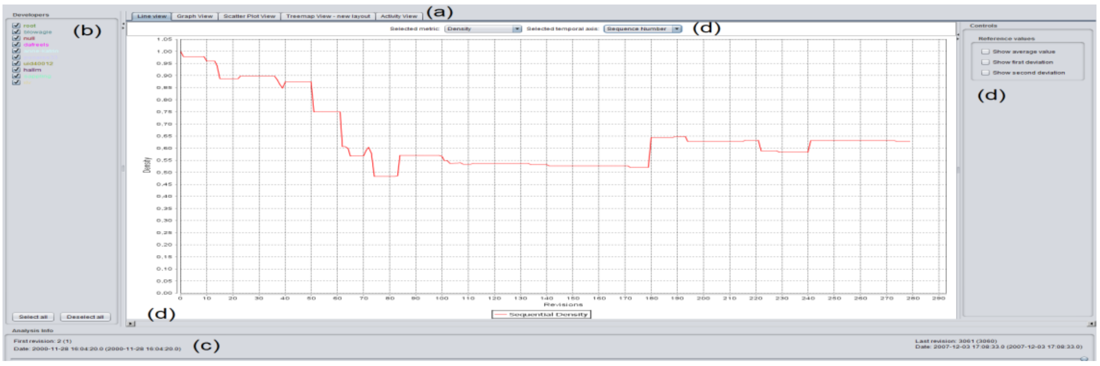


Figure 1. XFlow's simplified architectural model.

Four of the visualizations implemented were inherited from TransFlow, and will be just briefly mentioned. i) *Line Visualization* (Figure 2): presents a line chart that compares two variables along a two-dimensional Cartesian system. It enables researchers to identify trends and make predictions. ii) *Graph Visualization*: shows dependencies between any combination of XFlow entities (developers and artifacts) in a Prefuse graph structure [Heer *et al.* 2005], where each entity is represented as a vertex and the edges represent the dependencies between them (thickness indicates dependency degree). This visualization is particularly useful for the investigation of socio-technical networks of a software project (Section 4). iii) *Scatterplot Visualization* (Figure 3): shows correlation between variables by plotting a collection of points. Each point represents a commit, and receives a specific color for each developer. This visualization enables the investigation of the contributions made by a single developer (e.g., if they have reached the core files of the investigated system). iv) *TreeMap Visualization*: uses nested rectangles to display information with hierarchical characteristics. XFlow employs TreeMaps to present packages and artifacts (files) in a scalable manner, i.e. the tool is able to accurately portray the system’s whole structure at a given time and for specific developers (Section 4). For further details on these visualizations, please refer to [Costa *et al.* 2009].



First revision: 2 (1)
Date: 2000-11-28 16:04:20.0 (2000-11-28 16:04:20.0)

Last revision: 3061 (3060)
Date: 2007-12-03 17:08:33.0 (2007-12-03 17:08:33.0)

Figure 2. Line Visualization

The novelty amidst visualizations comes in the form of the *Activity Visualization* (Figure 4). This visualization is composed of a bar chart and a stacked area chart, both receiving colors to visually distinguish developers. The bar chart depicts the developers’ contribution period (since they entered the project until they left it), while the stacked area chart presents the degree of contribution made by those developers in the form of a metric (e.g., *number of commits per month* or *number of artifacts modified per week*), with the actual value of the metric being indicated by its area on the chart. This visualization enables users to quickly glance the activity history in a software project.

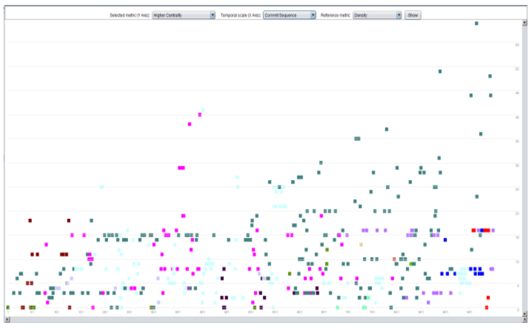


Figure 3. Scatterplot Visualization

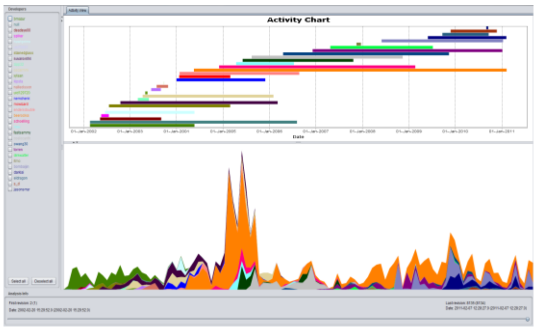


Figure 4. Activity Visualization

Finally, XFlow offers a set of filters and controls to customize the visualizations and help users to focus on specific details of a project. The *visualizations tabs* (Figure **2**a) aggregate tabs for all user-desired visualizations, and enable a user to quickly switch between visualizations. XFlow also offers common and specific controls to filter data displayed on visualizations. Common controls are filtering mechanisms shared by all visualizations, making possible to easily study a period of interest under the same conditions by forcing the same filtering parameters to all visualizations, being currently implemented in the form of the developers' pane and the analysis info bar. The *developers pane* (Figure 2B) is a common controller that lists all developers that contributed to project in the time interval specified during the analysis setup, and allows users to interact with visualizations by selecting which developers are relevant to the research being conducted, allowing further exploration of developers' evolution and their impact on the project. The *analysis info bar* (Figure 2C) is also a common controller, displaying general information about the analysis and contains a slider that enables users to focus on a given period of time. In each visualization there are three areas for *specific controls* (Figure 2D). These controls enable the user to further filter data, display references values, or simply choose an appropriate layout to the task in hand, available only for the currently observed visualization.

4. Using XFlow to Study Software Evolution

XFlow was designed to provide support to users interested in the comprehension of software evolution in an exploratory context. We expect that by means of analysis and comparison of different software systems, researchers will be able to formulate hypotheses and possibly identify patterns, practices and/or methodologies. In this section, we present usage scenarios for the tool, illustrating how it can be used to help researchers formulate and test hypothesis. The investigated projects are all open source software systems (FLOSS), namely: Apache Lucene, MegaMek, and jEdit.

Software Socio-Technical Evolution. XFlow's main feature is the ability to offer users visualizations that depict how a particular software system evolved over time and how each developer contributed to this system. There are numerous exploratory questions that can drive these studies, and so we will focus on a particular question: *how development teams deal with the departure of core developers?*

Although the whole software industry can benefit from learning how to handle with the departure of core developers, this notion is especially important for OSS projects due to the typical lack of formal commitment in this genre of software development. While for some OSS projects the departure of project's leaders compromises the entire project (e.g. the GIMP project, which suffered from a 20 month's hiatus [Ye & Kishida 2003]), we can see by MegaMek's Activity Visualization (Figure 4) that this particular project went through a smooth leadership transition (Figure 5A). The image, which was filtered to display visual data related to two developers only, reveals that while the project's founder and most active contributor (in terms of commits per month) was abandoning the project, a recently joined developer started to contribute. This new developer would later replace the founder not only in terms of contributions, but also as the project's announced lead developer.

To further observe this role exchange, we present MegaMek's TreeMap Visualization at the time of the founder's last contribution. The set of files changed by the project's founder (Figure 5B) and the to-be leader (Figure 5C) suggests that they've both shared knowledge about parts of the system (rounded square). Moreover, Figure 5D shows that the to-be leader evolved along with the project by increasing and spreading his contributions among diverse modules of the project.

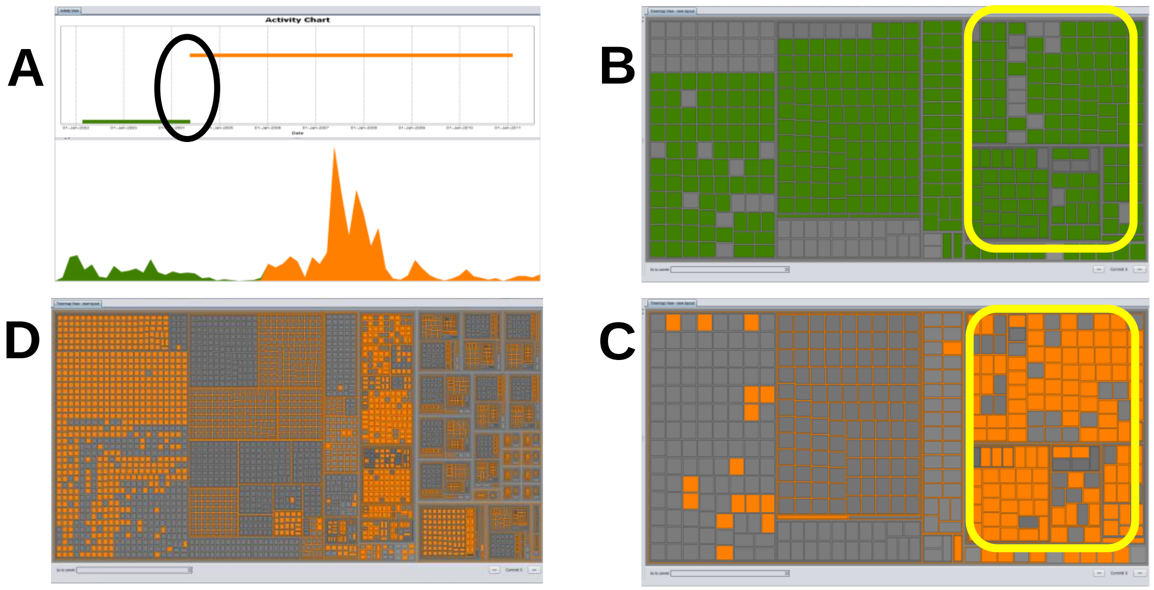


Figure 5. Project MegaMek analyzed from several perspectives.

Architectural Effects on Developers. XFlow provides several ways to investigate the correlation between software architecture and its effects among developers. By employing XFlow's Graph Visualization, we address the following question: *how a software system grows to accommodate new features?*

Figure 6 illustrates Apache Lucene artifacts dependency graph (A) and coordination requirements (B). It is clear that both are very dense, indicating not only a highly coupled architecture, but also a great need for coordination among projects developers. Consequently, all developers often work on the same set of files. Figure 7, in turn, illustrates jEdit artifacts dependency graph (A) and coordination requirements (B). In this case, the dependency graph is also dense (although "slightly" less than Lucene's), but interesting enough, its coordination requirements is shown as a disconnected graph. This suggests that jEdit developers are "experts" in specific modules or small set of files.

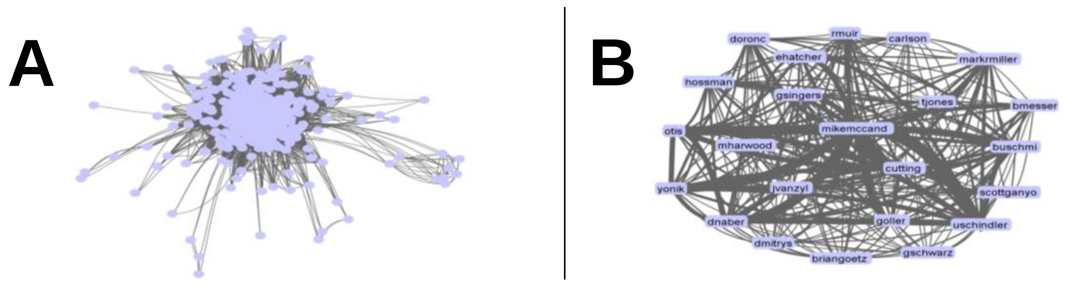


Figure 6. Apache Lucene Dependencies Graph

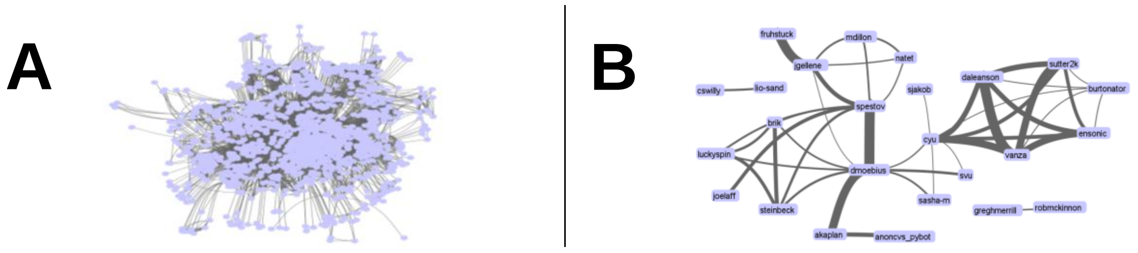


Figure 7. jEdit Dependencies Graph

We believe that further investigation of the difference between Lucene and jEdit may help software engineering researchers to understand the design of modular systems and their associated coordination needs.

5. Related Work

A number of previous studies on Software Evolution proposed several tools and approaches to understand how a software system evolved through time, including Augur [de Souza *et al.* 2005], CodeSaw [Gilber & Karahalios 2007], and Tesseract [Sarma *et al.* 2009]. Augur aims to make socio-technical structures visible by displaying software modules, developers and their relationship within the same frame, and has been used to identify core-periphery shifts in open source communities. CodeSaw aims to reveal social dynamics buried in software systems by analyzing source code artifacts and the project's communication that surrounds them, and is able to discover trends and roles amidst software developers by displaying collected data from configuration management repositories and mailing lists. Tesseract is a socio-technical dependency browser that, similarly to CodeSaw, analyzes different project archives to identify numerous relationships between developers in a project and display them on cross-linked views. While these tools offer means to correlate artifact-centric evolution aspects with software developers' evolution, they were not designed with this focus, thus not making it possible to have a full grasp at the relationship between these two views. Furthermore, although both Tesseract and CodeSaw are able to extract information from more repositories than XFlow, they provide limited support for structural analysis of software systems. In turn, our tool offers deeper insight possibilities by evaluating system size, calculating different coupling metrics, and supplying rich interactive visualizations.

Other important, but less related work, include the development of software evolution tools that focus exclusively on the technical or social perspective. The tools in the first category support the investigation of artifacts and source-code dependencies residing in software repositories aiming at of providing insights about the software structure and improving maintainability. Examples of tools in this category include eRose [Zimmerman *et al.* 2005] and Churrasco [D'Ambros & Lanza 2010]. Churrasco is a web-based framework that calculates software metrics and logical dependencies over collected data from VCSs, provides different visualizations to support the analysis of such measures, and supports collaboration through the sharing of user-created annotations. eRose is a plugin for the Eclipse IDE that aims to reduce errors due to incomplete modifications in files by mining a locally mirrored CVS repository and offering suggestions and predictions of files that are likely affected by ongoing changes.

The tools in the second category (social perspective) were designed to offer insights about how developers conducted the software development process, aiming to enhance the awareness level of developers or learn from social structures formed over their interactions with each other. Examples of these tools include Ariadne [Trainer *et al.* 2005] and RaisAware [Costa *et al.* 2008], both of them aiming to help developers accomplish their tasks by suggesting coordination requirements.

6. Conclusions and Future Work

In this paper we presented XFlow, an extensible tool whose main goal is to enable interactive investigation of a software project evolutionary process, thus supporting exploratory case studies. XFlow accomplishes such goal by collecting data from configuration management systems, processing it by computing metrics and, finally, presenting five powerful interactive visualizations. We have also presented illustrative usage scenarios involving the analysis of three FLOSS projects, highlighting the tool ability in providing insightful information for the formulation and test of hypothesis.

In addition to conducting studies of software evolution, we plan to extend XFlow in three directions: implementing connectors for others configuration management systems; finishing a structural dependencies identification module that is under development; and proposing and evaluating a new graphical user interface project.

Acknowledgments.

Francisco Santana and Marco Gerosa received individual grants from CNPq. In addition, the first and third authors were supported by the Brazilian Government under grant CNPq 473220/2008-3 and by the Fundação de Amparo à Pesquisa do Estado do Pará through "Edital Universal N° 003/2008".

References

Ball T., Kim Jung-Min, Porter A., Siy H. (1996) "If Your Version Control System Could Talk …" in Proc. Workshop on Process Modeling and Empirical Studies of Software Engineering, p. 33-43.

Bennett K.H. and Rajlich V.T. (2000), Software Maintenance and Evolution: a Roadmap, in Proceedings of the Conf. on The Future of Software Engineering, ICSE'00, Limerick, Ireland, ACM Press, pp. 75-87.

Bevan, J., Whitehead, J. E., Kim, S. Jr., and Godfrey, M. (2005) Facilitating Software Evolution Research With Kenyon. In: *SIGSOFT Softw. Eng. Notes* 177-186.

Bode S. (2009) "On the role of evolvability for architectural design," in Fischer, S.; Maehle, E.; Reischuk, R. (Hrsg.): INFORMATIK 2009 – Im Focus das Leben. Workshop Modellierung und Beherrschung der Komplexität, GI-Edition Lecture Notes in Informatics (LNI) p. 3256-3263.

Booch G. (2007) Object-Oriented Analysis and Design with Applications, 3rd ed.: Addison-Wesley.

Cataldo, M., Wagstrom, P., Herbsleb, J. D., and Carley, K.M. (2006) "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," In Proc. Conf. Computer Supported Cooperative Work, pp. 353-362.

Conway, M.E. (1968) “How Do Committees invent?”. In Datamation, 14 (4). pp. 28-31.

Costa, J., Santana, F., and de Souza, C. (2009) “Understanding open source developers' evolution using TransFlow”. In *Proceedings of the 15th international conference on Groupware: design, implementation, and use* (CRIWG'09).

Costa, J., Feitosa, R., and de Souza, C. (2008) “RaisAware: Uma Ferramenta de Auxílio À Engenharia de Software Colaborativa Baseada em Análises de Dependências”. In *Proceedings of the 2008 Simpósio Brasileiro de Sistemas Colaborativos* (SBSC '08).

D’Ambros M, Lanza M. (2010) “Distributed and Collaborative Software Evolution Analysis with Churrasco. *Science of Computer Programming*”. 75(4):276-287]

de Souza, C., Froehlich, J., and Dourish, Paul. (2005) “Seeking the source: software source code as a social and technical artifact”. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work* (GROUP '05). ACM, New York, NY, USA, 197-206.

Gall H, Hajek K, Jazayeri M. (1998) “Detection of logical coupling based on product release history”. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272).* p 190-198.

Gilbert, E., Karahalios, K.: CodeSaw (2007) “A Social Visualization of Distributed Software Development”. In: Human-Computer Interaction - INTERACT 2007, p. 303-316

Heer J., Card S., and Landay J. (2005) “Prefuse: a toolkit for interactive information visualization”. In: Proceedings of the 2005 SIGCHI conference on Human factors in computing systems.

Kagdi H, Collard M, Maletic J. (2007) “A survey and taxonomy of approaches for mining software repositories in the context of software evolution”. *Journal of Software Maintenance and Evolution: Research and Practice*. 19(2):77–131.

Koch, S. (2007). Software evolution in open source projects—a large-scale investigation. J. Softw. Maint. Evol. 19, 361-382.

Mens T, Demeyer S. (2008). *Software Evolution* (1 ed.). Springer Publishing Company, Incorporated.

Olatunji SO, Idrees SU, Al-Ghamdi YS, Al-Ghamdi JSA. (2010) Mining Software Repositories–A Comparative Analysis. *IJCSNS*. 10(8):161.

Parnas, D.L. (1972) “On the Criteria to be Used in Decomposing Systems into Modules” In Communications of the ACM, Volume 15 Issue 12, New York, USA, ACM Press.

Sarma, A., Maccherone, L., Wagstrom, P., and Herbsleb J. (2009) “Tesseract: Interactive Visual Exploration of Socio-Technical Relationships” In Software Development. In IEEE 31st International Conference on Software Engineering (ICSE’09)

Shneiderman, B. (1996) “The eyes have it: A task by data type taxonomy for information visualizations”. In Proceedings of 1996 IEEE Conference on Visual Languages, Boulder, CO.

Trainer, E.; Quirk, S.; de Souza, C., Redmiles, D. (2005) “Bridging the Gap between Technical and Social Dependencies with Ariadne”. In: Proceedings of the eclipse Technology eXchange (eTX) Workshop, San Diego, CA.

Yunwen Ye and Kouichi Kishida. (2003) “Toward an understanding of the motivation Open Source Software developers”. In *Proceedings of the 25th International Conference on Software Engineering* (ICSE '03). IEEE Computer Society, Washington, DC, USA, 419-429.

Zimmermann T., Zeller A., Weissgerber P., Diehl S. (2005) “Mining version histories to guide software changes”. *IEEE Transactions on Software Engineering*. 31(6):429-445.